Remarks

Applicant respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 1, 3-8, 10, 11, 17, 19 and 30-37 are pending in the application. Claims 1, 3-8, 10, 11, 17, 19 and 30-37 are rejected. Claims 2, 9, 12-16, 18, and 20-29 have previously been canceled without prejudice. No claims have been allowed. Claims 1, 7, 8, and 19 are independent.

Claim Objections

The Action objects to claim 3. The correction is reflected in claim 3 as amended.

Claim Rejections 35 USC § 101

The Action asserts a rejection of claims 1, 3-6, 17, and 30-31 as directed to a computer readable medium, which, it is asserted, may include communication medium. Applicants respectfully disagree with the Examiner's characterization of the claims and relevant law, and believes that the claims in their previous state satisfied 35 USC § 101. Nevertheless, Applicants have amended the above-rejected claims in an effort to expedite prosecution. Specifically, the language "A computer-readable storage medium" has been added to the beginning of the independent claims as kindly suggested by the Examiner.

Claim Rejections 35 USC § 103(a)

The Action asserts a rejection of claims 1, 3-8, 10, 11, 17, 19, and 30-37 as being unpatentable over U.S. Patent No. 5,937,186 to Horiguchi et al. (Horiguchi) in view of U.S. Patent No. 6,708,194 to Korn (Korn). Applicants respectfully disagree, and traverse. To establish prima facie obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art. In re Royka, 490 F.2d 981, 180 USPQ 580 (CCPA 1974). "All the words in a claim must be considered in judging the patentability of that claim against the prior art." In re Wilson, 424 F.2d 1382, 1385, 165 USPQ 494, 496 (CCPA 1970).

Horiguchi cannot be combined with Korn.

The Action asserts that Korn's teaching can be combined with Horiguchi's teaching. The motivation given is that "one would have been motivated to do so to handle primary (parent) thread (parent) and other (child) threads when parallel executing as suggested by Korn (e.g., col. 3: 47- col. 4:14.) Applicants respectfully disagree. To combine references, there must be some expectation of some advantage. MPEP 2144.IV. However, "the proposed modification cannot render the prior art unsatisfactory for its intended purpose." MPEP 2143.01.V. Also, The proposed medication cannot change the principle of operation of a reference. MPEP 2143.01.V1.

Further, the MPEP requires explicit analysis, and not conclusory statements. As MPEP § 2142 states: The key to supporting any rejection under 35 U.S.C. 103 is the clear articulation of the reason(s) why the claimed invention would have been obvious. The Supreme Court in KSR International Co. v. Teleflex Inc., 550 U.S. ___, ___, 82 USPQ2d 1385, 1396 (2007) noted that the analysis supporting a rejection under 35 U.S.C. 103 should be made explicit. The Federal Circuit has stated that "rejections on obviousness cannot be sustained with mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." In re Kahn, 441 F.3d 977, 988, 78 USPQ2d 1329, 1336 (Fed. Cir. 2006). See also KSR, 550 U.S. at ___ , 82 USPQ2d at 1396 (quoting Federal Circuit statement with approval).

The two references that the Action purports to combine are so different that combination makes no sense at all, and so would certainly render Horiguchi unsatisfactory for its intended purpose, if combined with Korn, and would also change the principle of operation of Horiguchi, to say the least.

Horiguchi is concerned with "processing an asynchronous interrupt of a processing entity." [Horiguchi, Abstract.] The processing runs strictly on a single (multi-threaded) operating system. By contrast, Korn is concerned with porting processes that run on single-thread operating systems to those that run on multi-threaded operating systems. [Korn, abstract.] As Horiguchi has nothing to do with porting one operating system portion to another, it is nonsensical to place a feature for porting an operating system portion into a patent that does no such thing. Further, as to the suggested improvement, it is unclear, to say the least, why "handing primary (parent) threads and other (child) threads" (the improvement suggested by the Action, pp. 5-6) would produce any sort of improvement to Horiguchi. Horiguchi does not even discuss parent or

children threads, let alone any sort of parent-child relationships. Shoehorning in a parent-child thread relationship into a patent that has no such constructs, nor any use for such constructs would do nothing but require additional constructs with no apparent purpose, making Horiguchi unsatisfactory for its intended purpose, as it would either not work, or at a very best case scenario, be unnecessarily complicated for no perceived benefit. As there is no improvement suggested, nor any improvement even envisioned, the statement, on its face, is conclusory, as is not allowed. For at least this reason Horiguchi cannot be combined with Korn.

To reach a proper determination under 35 U.S.C. 103, the examiner must step backward in time and into the shoes worn by the hypothetical "person of ordinary skill in the art" when the invention was unknown and just before it was made. In view of all factual information, the examiner must then make a determination whether the claimed invention "as a whole" would have been obvious at that time to that person. Knowledge of applicant's disclosure must be put aside in reaching this determination, yet kept in mind in order to determine the "differences," conduct the search and evaluate the "subject matter as a whole" of the invention. The tendency to resort to "hindsight" based upon applicant's disclosure is often difficult to avoid due to the very nature of the examination process. However, impermissible hindsight must be avoided and the legal conclusion must be reached on the basis of the facts gleaned from the prior art.

As only a conclusory statement was provided to combine Horiguchi with Korn, it appears that impermissible hindsight was used to make the determination that these two references could be combined. Separately, Horiguchi cannot be combined with Korn because the Korn techniques to port one type of operating system to another have nothing to with Horiguchi's techniques to process asynchronous interrupts. For at least the above reasons, Applicants respectfully submit that the two references cannot be combined, and as such, the Action has failed to establish a prima facie case of obviousness on the basis of these two references.

Claim 1.

Neither Horiguchi nor Korn, either together or separately, teach or suggest, e.g., the claim 1 language "wherein, after the fork method creates the third state frame, value of a variable of the executing program can be accessed by checking, in backwards order that the frames were created, for the value of the variable in the respective frames."

The Action concedes that Horiguchi does not teach or suggest the above language. [Action, page 5.] Nonetheless, the Action contends that Korn so discloses. Applicants respectfully disagree.

To teach or suggest, the Action cites to five separate locations in Korn. Each will be discussed in turn.

Korn, 6: 32-46, shown below:

> The semantics of POSIX processes are determined by the above characteristics and also by a set of operating system functions which POSIX defines for processes: fork and the exec family of functions. fork is the fiction in POSIX which creates a new process. When fork is executed by a process, the function creates a new process which is a child of the process that executed fork; the new process has its own process id, and its parent id is that of the process that created it, but the new process otherwise receives a copy of the context 103 of its parent process. The new process is then started. Since the new process has a copy of the context 103 of its parent, both processes continue executing fork. The function returns the pid of the child to the parent and 0 to the child.

This describes a fork which creates a new process that "receives a copy of the context 103 of its parent process." The plain reading of the text, above, is that the entire context is received as a single copy, teaching away from the necessity of locating the value of any given variable, let alone checking in a for the value in any specific manner. All this teaches against the language of claim 1 "value of a variable of the executing program can be accessed by checking, in backwards order that the frames were created, for the value of the variable in the respective frames...."

Korn, 10:54-63, shown below:

> Other fields of interest are inexec 437, which is set when the POSIX process is executing an exec function, state 439, which indicates the POSIX process's current state, time data 441, which includes data that permits a POSIX process to set an alarm for a future time and suspend itself until the time is reached and data indicating how much CPU time and elapsed time the process has required. *forksp 443, finally, is a pointer to the top of the stack of the parent POSIX process at the time a child POSIX process is spawned. Its use will be explained in the discussion of fork which follows.

This describes a POSIX process suspending itself until a future time. When the future time is reached, "a child POSIX process is spawned." This passage, also does not discuss any specific way of checking for the value of a variable in the spawned process, and as such does not

teach or suggest a specific method of checking for the value of a variable, does not mention frames, does not mention checking anything in backwards order, and so does not suggest the claim 1 language "wherein, after the fork method creates the third state frame, value of a variable of the executing program can be accessed by checking, in backwards order that the frames were created, for the value of the variable in the respective frames."

Korn, 11:10-23, shown below:

> In the preferred embodiment, a new Win32 process results whenever a POSIX process executes a fork function or a exec function. In the first case, the new Win32 process corresponds to a new POSIX process which is the child of the POSIX process executing fork; in the second case, the new Win32 process replaces the Win32 process which formerly corresponded to the POSIX process executing exec. With both fork and exec, the relationship between the POSIX process which created the new Win32 process and the new Win32 process are handled by the wait thread in the Win32 process which corresponds to the POSIX process that is executing the fork or exec function. The wait thread for the Win32 process maintains data structures which keep track of the children of the corresponding POSIX process; the wait thread further ensures that when a child POSIX process is terminated, the parent receives a SIGCHLD signal.

This passage describes two different ways for a WIN32 operating system to replicate a POSIX operating system; a *fork*, which creates a new Win32 process that exists alongside the original POSIX operating system process; and an *exec*, which replaces a Win32 process which originally corresponded to a process in the original POSIX operating system. This does describe a wait thread "that maintains data structures which keep track of the children of the corresponding POSIX process." However, like the above quotes from Korn, this passage, also, does not discuss checking for the value of a variable in the child process; rather, the process is "replicated," and as such does not teach or suggest a **specific method** of checking for the value of a variable, does not mention frames, does not mention checking anything in **backwards order**, and so does not suggest the claim 1 language "wherein, after the fork method creates the third state frame, value of a variable of the executing program can be accessed by checking, in backwards order that the frames were created, for the value of the variable in the respective frames."

Korn, 13:62-14:35, shown below:

> As described above, when a first POSIX process executes a fork function, the result is a second POSIX process that is a child of the first POSIX process and has

> a copy of all of the state of its parent. IN consequence, at the end of the fork function, both the parent and the child are executing the same code, each, of course, in its own address space. An implementation of fork in a port of the POSIX operating system to the Win32 operating system must solve two problems: establishing the parent-child relationship, which has no equivalent in Win32, and making the copy of the parent's state for the child.
>
> FIG. 7 provides an overview of fork in a preferred embodiment. As shown in FIG. 7, each POSIX process 704 includes a pproc_t entry 702 for the POSIX process and a Win32 process 703 corresponding to the POSIX process. Win32 process 703 has associated with it process state 705. FIG. 7 shows at 701 a POSIX process 704(a) prior to execution of fork. Process 704(a) has pproc_t entry 702(a), Win32 process 703(a), and process state 705. At 706, FIG. 7 shows the result of an execution of fork by POSIX process 704(a). POSIX process 704(a) now has a child POSIX process 704(b). POSIX process 704(b) has its own pproc_t entry 702(b), its own Win32 process 703(b), and a copy 707 of state 705 belonging to Win32 process 703(a). The parent-child relationship between POSIX process 704(a) and POSIX process 704(b) is of course indicated by the field ppid 433 in pproc_t entry 702(a), and is represented in FIG. 7 by arrow 708.
>
> In the following discussion, the POSIX process which invokes fork will be called the parent and the POSIX process which results from the execution of fork will be called the child. In the preferred embodiment, fork is implemented as follows: When the parent POSIX process invokes fork, the primary thread of the corresponding Win32 process (the parent Win32 process) executes a setjmp function which established an environment that a longjmp function executed by the primary thread of the Win32 process corresponding to the child POSIX process (the child Win32 process) can return to. Of course, what the child Win32 process returns to is the copy of the environment in the state of the child POSIX process.

This extended passage describes porting a POSIX process to a Win32 process, and the changes that are required. Essentially, a POSIX fork process starts a process in the Win32 operating system, and is provided a location in the Win32 Operating system in which to return. This passage does not discuss how the child processes are created at all, merely how they behave once so created. Therefore, this passage, most certainly does not discuss any way of checking for the value of a variable in the child process, and as such does not teach or suggest *a specific method* of checking for the value of a variable, does not mention frames, does not mention checking anything in **backwards order**, and so does not suggest the claim 1 language "wherein, after the fork method creates the third state frame, value of a variable of the executing program

can be accessed by checking, in backwards order that the frames were created, for the value of the variable in the respective frames."

Moreover, Horiguchi cannot be combined with Korn, as shown above.

For at least these reasons, claim 1 is in condition for allowance.

Accordingly, favorable reconsideration and withdrawal of the rejection of independent claim 1 under 35 U.S.C. § 103 are respectfully requested.

In the event that the Office maintains the rejection of claim 1 under 35 U.S.C. §103, Applicant respectfully requests that the Office, in the interests of compact prosecution, identify on the record and with specificity sufficient to support a prima facie case of anticipation, where in the Horiguchi or the Korn patent the subject feature of claim 1 of "wherein, after the fork method creates the third state frame, value of a variable of the executing program can be accessed by checking, in backwards order that the frames were created, for the value of the variable in the respective frames." is alleged to be taught. Specifically, Applicant respectfully requests the subject features "checking in backwards order" are identified.

Claims 3-6 and 30-31.

Additionally, claims 3-6 and 30-31 depend from claim 1. In the interest of brevity, Applicants do not belabor the language of each of the dependent claims, but point out that they recite novel and nonobvious features allowable over the Horiguchi - Korn Combination. Further, since they depend from claim 1, they should be allowed for at least the reasons stated for claim 1. Claims 3-6 and 30-31 should be allowable for at least the reasons given. Such action is respectfully requested.

Claim 7.

Claim 7 reads:

> 7. (Previously Presented) A computerized method comprising:
> receiving via an application programming interface a request to create a state save;
> in response to the request, saving a first representation of a state of an executing program comprising copying state of the program required to return to the moment the state was saved as a first state frame;
> creating a blank state frame with a backward link to the first state frame as a current state frame;

maintaining a second representation of subsequent state comprising changes made to the state of the executing program after the first representation in the current state frame;

in response to a request for value of a variable after the request to create a state save, checking for the value of the variable in the first state frame; and

changing the current state frame to the first state frame upon receiving a state set request at the application programming interface.

Neither Horiguchi nor Korn, either together or separately teach or suggest, e.g., the claim 7 language "maintaining a second representation of subsequent state comprising changes made to the state of the executing program after the first representation in the current state frame...."

To teach or suggest, the action cites to Horiguchi, 7:9-53, quoted below.

FIG. 4 depicts a stack frame 402 according to a preferred embodiment of the present invention. The stack frame 402 includes a segment having a predetermined, fixed length (this segment is called the fixed part 404) and a segment whose length varies from function to function (this segment is called the variable part 406).

The fixed part 404 includes a backward stack frame pointer field 408 and a forward stack frame pointer field 410. These fields 408 and 410 are used to chain the stack frame 402 to the two stack frames in the invocation stack that respectively precede and follow the stack frame 402. In particular, the backward stack frame pointer field 408 stores the starting address (called the backward pointer) of the stack frame which precedes the stack frame 402 in the invocation stack. The forward stack frame pointer field 410 stores the starting address (called the forward pointer) of the stack frame which follows the stack frame 402 in the invocation stack.

The fixed part 404 also includes a register save area (RSA) 412 which is used to store information pertaining to the state of the function (referred to as the calling function) associated with the stack frame 402 when the calling function invokes another function (referred to as the called function). Specifically, when the calling function invokes the called function, the called function's prologue portion stores information pertaining to the state of the calling function in the register save area 412 of the fixed part 404. The function state information which is stored in the register save area 412 includes data contained in one or more of the registers 611 that were being used by the calling function, as well as the return address of the calling function.

The fixed part 404 also includes a next available byte (NAB) pointer field 414 which stores the address (called the next available byte pointer) of the byte in the invocation stack that immediately follows the stack frame 402 (more particularly,

> the byte in the invocation stack that immediately follows the variable part 406 of the stack frame 402). Note that the NAB pointer field 414 essentially performs the same function as the forward stack frame pointer field 410. Thus, in one embodiment of the present invention, the forward stack frame pointer field 410 is not used.

The rather extensive section of Horiguchi cited, above, discusses a stack frame that is used to store "information pertaining to the state of the calling function" which, itself is "data contained in one or more of the registers 611 that were being used by the calling function, as well as the return address of the calling function." [*Id.*] Also stored is address of the next stack frame. ["The fixed part 404 also includes a next available byte (NAB) pointer field 414 which stores the address (called the next available byte pointer) of the byte in the invocation stack that immediately follows the stack frame 402. *Id.*] That is, the stack frame stores information used by **a single function**. Storing information relating to a single function is not "maintaining a second representation of subsequent state comprising changes made to the state of the executing program after the first representation in the current state frame...." as there is no representation of subsequent state," rather, the information stored ("data contained in one or more of the registers 611 that were being used by the calling function, as well as the return address of the calling function," *id.*), is **current** state information, used to store the state of a current function, not "a second representation of subsequent state...." For at least this reason, claim 7 is in condition for allowance.

As a separate reason for allowability, neither Horiguchi nor Korn, either together or separately teach or suggest, e.g., the claim 7 language "creating a blank state frame with a backward link to the first state frame as a current state frame...."

The Action concedes that Horiguchi does not teach or suggest "creating a blank state frame with a backward link to the first state frame as a current state frame." [Action, page 5.] Nonetheless, the action contends that Korn so discloses. Applicants respectfully disagree.

To teach or suggest, the Action cites to Korn, 6:33-54 and Korn 7:63 – 8:9, reproduced below.

> The semantics of POSIX processes are determined by the above characteristics and also by a set of operating system functions which POSIX defines for processes: fork and the exec family of functions. fork is the fiction in POSIX which creates a new process. When fork is executed by a process, the function creates a new process which is a child of the process that executed fork; the new

> process has its own process id, and its parent id is that of the process that created it, but the new process otherwise receives a copy of the context 103 of its parent process. The new process is then started. Since the new process has a copy of the context 103 of its parent, both processes continue executing fork. The function returns the pid of the child to the parent and 0 to the child.
>
> When a POSIX process executes one of the exec family of functions, the result is that the process begins executing a program specified as an argument to the invocation of the exec function. In terms of process context 103, program text 105 is replaced by the program specified in the call to exec, stack 114 is replaced by a new stack, and machine registers 109 are replaced by a set of machine register values which arc proper for the new program and the new stack. [Korn, 6:33-54.]
>
> As would be expected from the foregoing, there are also no functions in Win32 corresponding to the POSXI fork and exec functions. Instead, there are CreateProcess and CreateThread functions. CreateProcess simply creates a new Win32 process with a single thread, called the primary thread. An argument in the invocation specifies a portion of the process's code which is to be executed by the primary thread. The new process can inherit handles, environmental variables, the current directory, and the console of the parent process, but is otherwise completely independent. In contrast to a POSIX process created by fork, the new process does not know the identity of the process that created it and does not share the parent's code or have copies of the parent's stack and machine registers. [Korn 7:63 – 8:9]

Korn describes porting the POSIX operating system to a WIN32 operating system. [Korn, Abstract.] The first Korn quote, [Korn, 6:33-54] describes the behavior of the POXIS operating system prior to any port to the WIN32 operating system. The second quote [Korn 7:63 – 8:9] describes the WIN 32 Operating system. Thus the two quotes describe different systems, which, are otherwise incompatible—a POSIX process (first reference used by the examiner, e.g., Korn col. 6, 33-54) will be implemented using a Win32 system (second reference, e.g., Korn col. 8-40.) However, the Action treats the two quotes as belonging to the same process, or at least the same general set of actions, as shown by the quote, from the Action, page 7, below.

> However, Korn further discloses:
>
> *creating a blank state frame with a backward link to the first state frame as a current state frame* (e.g., col.6: 33-54, generating a new (empty) frame in the forked (child) process/thread and after that, copying state/context of forking (parent) process/thread to said new frame; col.7: 63 – col.8: 9); *and*

As the two separate Korn references cited belong to different systems (POSIX and Win32), they cannot be combined to describe a single action, e.g., Korn teaches away from "creating a blank state frame with a backward link to the first state frame as a current state frame;" as, at least, the initial representation in the POSIX system is quite incompatible with the Win32 system, as thoroughly documented by Korn at col 6, line 55 to col. 8, line 40.

Additionally, Horiguchi cannot be combined with Korn, as shown above.

For at least these reasons, claim 7 is in condition for allowance.

Accordingly, favorable reconsideration and withdrawal of the rejection of independent claim 7 under 35 U.S.C. § 103 are respectfully requested.

Claims 17, 32-34.

Additionally, claims 17, and 32-34 depend from claim 7. In the interest of brevity, Applicants do not belabor the language of each of the dependent claims, but points out that they recite novel and nonobvious features allowable over the Horiguchi - Korn combination. Further, since they depend from claim 7, they should be allowed for at least the reasons stated for claim 7. Claims 17, and 32 - 34 should be allowable for at least the reasons given. Such action is respectfully requested.

Claim 8.

Independent claim 8 recites in part: "the program including a method for locating a value updated during the prior evolving present state and not present in the second state frame by following a back pointer from the second state frame to the first state frame, reading location value from the first state frame and storing the location value in the second state frame." Neither Horiguchi nor Korn separately or in combination teaches or suggests the claim 8 language, above.

The action concedes that "Horiguchi does not explicitly disclose a first state frame comprising an initial representation of a prior evolving present state of the program" [Action, page 8,] but then says that Korn does so disclose. Applicants disagree and traverse.
A "prior evolving present state" is described, e.g., in the Specification at page 4, line 25 to line 5, line 2, quoted below.

> Notice that there are at least two states being discussed here. A first state is an evolving present state of a program or model. It is the evolving state of a program

> or model that is defined or referenced as a first class citizen via an imperative program construct or paradigm. Once this program model or state is defined and referenced, it is saved in a data structure and can be accessed as a first class citizen from the program or model.

To teach or suggest "a first state frame comprising an initial representation of a prior evolving present state of the program" the Action cites to Korn col. 6, lines 33-54, and col. 7:63 to col. 8:9, quoted below.

> The semantics of POSIX processes are determined by the above characteristics and also by a set of operating system functions which POSIX defines for processes: fork and the exec family of functions. fork is the fiction in POSIX which creates a new process. When fork is executed by a process, the function creates a new process which is a child of the process that executed fork; the new process has its own process id, and its parent id is that of the process that created it, but the new process otherwise receives a copy of the context 103 of its parent process. The new process is then started. Since the new process has a copy of the context 103 of its parent, both processes continue executing fork. The function returns the pid of the child to the parent and 0 to the child.
>
> When a POSIX process executes one of the exec family of functions, the result is that the process begins executing a program specified as an argument to the invocation of the exec function. In terms of process context 103, program text 105 is replaced by the program specified in the call to exec, stack 114 is replaced by a new stack, and machine registers 109 are replaced by a set of machine register values which arc proper for the new program and the new stack. [Korn col. 6, lines 33-54.]
>
> As would be expected from the foregoing, there are also no functions in Win32 corresponding to the POSXI fork and exec functions. Instead, there are CreateProcess and CreateThread functions. CreateProcess simply creates a new Win32 process with a single thread, called the primary thread. An argument in the invocation specifies a portion of the process's code which is to be executed by the primary thread. The new process can inherit handles, environmental variables, the current directory, and the console of the parent process, but is otherwise completely independent. In contrast to a POSIX process created by fork, the new process does not know the identity of the process that created it and does not share the parent's code or have copies of the parent's stack and machine registers. [Korn 7:63 – 8:9]

Korn describes porting the POSIX operating system to a WIN32 operating system. [Korn, Abstract.] The first Korn quote [Korn, 6:33-54] describes the behavior of the POXIS operating system prior to any port to the Win32 operating system. The second quote [Korn 7:63

– 8:9] describes the WIN 32 Operating system. Thus the two quotes describe different systems, which, are otherwise incompatible—a POSIX process (first reference used by the examiner, e.g., Korn col. 6, 33-54) will be implemented using a Win32 system (second reference, e.g., Korn col. 8-40.) However, the Action treats the two quotes as belonging to the same process, or at least the same general set of actions, as shown by the quote, from the Action, page 7, below.

As the two separate Korn references cited describe behavior of different operating systems (POSIX and Win32), they cannot be combined to describe a single action, e.g., Korn strongly teaches away from "a first state frame (e.g., presumably from the POSIX operating system) comprising an initial representation of a prior evolving present state of the program (e.g., presumably from the Win32 system)," as, at least, the initial representation in the POSIX system is quite incompatible with the Win32 system, as well-documented by Korn at, e.g., col 6, line 55 to col. 8, line 40.

Moreover, Horiguchi cannot be combined with Korn, as shown above.

Accordingly, favorable reconsideration and withdrawal of the rejection of claim 8 under 35 U.S.C. §103 is respectfully requested.

Claims 10-11.

Additionally, claims 10-11 depend from claim 8. In the interest of brevity, Applicants do not belabor the language of each of the dependent claims, but points out that they recite novel and nonobvious features allowable over the Horiguchi - Korn combination. Further, since they depend from claim 8, they should be allowed for at least the reasons stated for claim 8. Claims 10-11 should be allowable for at least the reasons given. Such action is respectfully requested.

Claim 19.

Claim19 recites:

> 19. A computerized method comprising computer executable instructions for performing a method comprising:
>
> receiving a request from a method, which takes as a parameter a state object, to create a saved state of an executing model;
>
> saving a first representation of a state of the executing model as a first state frame;
>
> saving a first representation of the state frame as the state object;
>
> creating a blank state frame with a backward link to the first state frame as a second state frame;

> maintaining, in the second state frame, a second representation of state changes made by the executing model after the first representation as the state changes occur; and
>
> reinstating the executing model state to the state of the first representation using the state object. [Emphasis added.]

Not to belabor the point, but using the same reasoning as found in claims 1 and 7, it can be seen that the Horiguchi-Korn combination does not anticipate claim 19.

Accordingly, favorable reconsideration and withdrawal of the rejection of independent claim 19 under 35 U.S.C. § 103(a) are respectfully requested.

Claims 35-37.

Additionally, claims 35-37 depend from claim 19. In the interest of brevity, Applicants do not belabor the language of each of the dependent claims, but points out that they recite novel and nonobvious features allowable over the Horiguchi - Korn combination. Further, since they depend from claim 19, they should be allowed for at least the reasons stated for claim 19. Claims 35-37 should be allowable for at least the reasons given. Such action is respectfully requested.

Formal Request For Interview

Upon reviewing this response, if any issues remain, the Examiner is formally requested to contact the undersigned prior to issuance of the next Office Action in order to arrange a telephonic interview. It is believed that a brief discussion of the merits of the present application may expedite prosecution. Applicants submit the foregoing formal Response so that the Examiner may fully evaluate Applicants' position, thereby enabling the interview to be more focused. This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

Conclusion

Claims 1, 3-8, 10, 11, 17, 19 and 30-37 should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By /Genie Lyons/
Genie Lyons
Registration No. 43,841